DATA TO IMPLEMENT DIVISION ALGORITHMS

To Implement slow and fast division algorithms, we need to gather some data related to their performance. The main data’s that are needed for the execution of division algorithms are,

1. INPUT DATA:

* Dividend: The number being divided.
* Divisor: The number dividing the dividend.

2. PROCESS IMPLEMENTATION TIME:

* Calculate the time taken by each algorithm to perform the division operation.
* Use a reliable timing mechanism or a profiler instrument to obtain accurate execution times which helps to find the efficiency of the algorithm.
* Repeat the division operation multiple times for each algorithm to calculate an average execution time.

3. NUMBER OF OPERATIONS:

* Count the total number of arithmetic operations (additions, subtractions, multiplications) performed and number of iterations to get the outcome of the division by each algorithm during the division process.
* This data helps understand the computational complexity of the algorithms.

4. DIVISION ACCURACY AND EXCEPTION HANDLING:

* Verify the accuracy of the division results obtained from each algorithm.
* Compare the quotient and remainder generated by each algorithm with the expected values.
* Ensure that the algorithms handle edge cases or exceptions correctly, such as division by zero or negative numbers.

5. ALGORITHMIC COMPLEXITY:

* Analyze the complexity of the restoring and goldschmidt’s division algorithms.
* Assess the time complexity (e.g., Big O notation) of each algorithm to determine their efficiency.

6. RESOURCE UTILIZATION:

* Measure the memory usage of each algorithm.
* Determine the amount of memory required by each algorithm during the division process.
* Compare the memory usage of the slow and fast division algorithms.

By collecting and analyzing the above data, we can make a comprehensive comparison between the slow and fast division algorithms as well as calculating efficiency of the division carried out by the processor of the device, evaluating factors such as execution time, computational complexity, accuracy, and resource utilization.

# DATA REQUIRED FOR RESTORING ALGORITHM:

To perform the restoring division algorithm, we need the following data:

1. Dividend: The number being divided.

2. Divisor: The number dividing the dividend.

3. Quotient: The result of the division operation.

4. Remainder: The value remaining after each division step.

5. Dividend length: The number of bits or digits in the dividend.

6. Divisor length: The number of bits or digits in the divisor.

7. Accumulator: A register used to hold the intermediate values during the division process.

8. Subtractor: A register used for subtraction operations during the division process.

9. Clock or counter: A mechanism to control the number of division steps.

These are the essential data elements needed to perform the restoring division algorithm. The algorithm involves a series of subtraction and shifting operations, which are repeated until the entire dividend is divided. The quotient and remainder are updated at each step, and the final quotient represents the result of the division.

## EXAMPLE:

INPUT DATA AND USAGE OF DATA:

Dividend: 101010 (42 in decimal)

Divisor: 11 (3 in decimal)

/Step 1:

Start with the dividend and divisor in long division format

101010 (Dividend)

÷ 11 (Divisor)

Step 2:

Compare the leftmost digit/group of digits in the dividend with the divisor

101010 (Dividend)

÷ 11 (Divisor)

// The leftmost digit is 10, which is greater than the divisor

Step 3:

Determine the largest multiple of the divisor that can be subtracted from the dividend

101010 (Dividend)

÷ 11 (Divisor)

// The largest multiple of 11 that can be subtracted from 101 is 11 (1 x 11)

// Subtract 11 x 1 from the dividend

101010 (Dividend)

- 11 (Divisor)

Step 4:

Bring down the next digit from the dividend

011010 (Dividend)

÷ 11 (Divisor)

Step 5:

Compare the leftmost digit/group of digits in the new dividend with the divisor

011010 (Dividend)

÷ 11 (Divisor)

// The leftmost digit is 01, which is greater than the divisor

Step 6:

Determine the largest multiple of the divisor that can be subtracted from the new dividend

011010 (Dividend)

÷ 11 (Divisor)

// The largest multiple of 11 that can be subtracted from 11 is 11 (1 x 11)

// Subtract 11 x 1 from the dividend

011010 (Dividend)

- 11 (Divisor)

// Since there are no more digits in the dividend, we stop the algorithm

PROCESS IMPLEMENTATION TIME:

Restoring algorithm takes O(n) time for the execution of division process.The value of n depends upon the number of iterations.

NUMBER OF OPERATIONS:

Restoring division algorithm involves the operations such as iterations ,division, shift, subtraction, comparing data etc…

ACCURATE OUTPUT DATA:

Quotient: 10 (2 in decimal)

Remainder: 000000 (0 in decimal)

RESOURCE UTILIZATION:

The memory needed for performing restoring algorithm depends on the size of the input datas.

The main memory requirement is for storing the dividend, divisor, and quotient during the division process. Typically, these values are represented using binary numbers, and the size of the memory required depends on the number of bits in each value.

In addition to the dividend, divisor, and quotient, the algorithm may also require temporary variables to store intermediate results during the computation. These temporary variables may include remainders, partial quotients, and carry/borrow flags.

The total memory needed can be calculated as follows:

1. Dividend: n bits (where n is the number of bits in the dividend).

2. Divisor: m bits (where m is the number of bits in the divisor).

3. Quotient: n bits (assuming the quotient has the same number of bits as the dividend).

In addition to the above, the algorithm may require some additional temporary variables to perform the subtraction and shifting operations. The exact number and size of these temporary variables depend on the specific implementation of the algorithm.

Therefore, the memory needed for the restoring division algorithm can be approximated as the sum of the sizes of the dividend, divisor, quotient, and any additional temporary variables required by the implementation.

# DATA REQUIRED FOR GOLDSCHMIDT’S ALGORITHM:

To perform the Goldschmidt division algorithm, we will need the following data:

1. Dividend: The number being divided.

2. Divisor: The number dividing the dividend.

3. Quotient: The result of the division operation.

4. Iterations: The number of iterations or steps to be performed in the algorithm.

5. Accumulator: A register used to hold the intermediate values during the division process.

6. Multiplier: A value used for multiplication operations.

7. Shift counter: A mechanism to track the number of right shifts performed during the algorithm.

8. Precision: The desired precision or number of significant digits in the quotient.

The Goldschmidt division algorithm is an iterative method that converges to the quotient by repeatedly refining an approximation. It involves multiplication, division, and shifting operations to improve the accuracy of the quotient with each iteration. The algorithm terminates after the specified number of iterations or when the desired precision is achieved.

EXAMPLE:

INPUT DATA AND ITS USAGE IN ALGORITHM:

Dividend: 101010 (42 in decimal)

Divisor: 11 (3 in decimal)

Step 1:

Align the divisor and dividend

101010 (Dividend)

÷ 11 (Divisor)

Step 2:

Initialize the quotient and remainder to zero

000000 (Quotient)

0000 (Remainder)

Step 3:

Compare dividend with divisor

101010 (Dividend)

÷ 11 (Divisor)

// If dividend >= divisor, subtract divisor from dividend

101010 (Dividend)

11 (Divisor)

Step 4:

Shift both the quotient and dividend one bit to the left

001000 (Remainder, shifted left)

000010 (Quotient, shifted left)

// Repeat steps 3 and 4

001000 (Dividend)

÷ 11 (Divisor)

001000 (Dividend)

11 (Divisor)

Step 4:

Shift both the quotient and dividend one bit to the left

001000 (Remainder, shifted left)

000010 (Quotient, shifted left)

// Repeat steps 3 and 4

001000 (Dividend)

÷ 11 (Divisor)

001000 (Dividend)

11 (Divisor)

// Since the dividend has become zero, we stop the algorithm

PROCESS IMPLEMENTATION TIME:

Goldschmidt division algorithm time complexity depends upon the efficiency of floating point arithmetic operations on the execution of the Verilog code and the additional computations applied.

NUMBER OF OPERATIONS:

Goldschmidt division algorithm involves the operations such as iterations(repition) ,division, multiplication, subtraction, comparing data ,Normalizaion, Initialization ,updation etc…

ACCURATE OUTPUT DATA:

Quotient: 000111 (7 in decimal)

Remainder: 000000 (0 in decimal)

RESOURCE UTILIZATION:

The memory requirements for the Goldschmidt division algorithm depend on several factors, including the precision of the numbers involved (number of digits or bits), the data types used to represent the numbers, and the specific implementation details.

The main memory requirements for the Goldschmidt division algorithm are as follows:

1. Input Numbers:

The algorithm requires memory to store the dividend and divisor, which are the input numbers for the division operation. The memory size will depend on the precision and data type used to represent these numbers.

2. Intermediate Variables:

During the iteration process, the algorithm uses intermediate variables to store the current reciprocal approximation, the product of the dividend and reciprocal, and the updated reciprocal approximation. The memory size for these intermediate variables will depend on the precision and data type used.

3. Additional Variables:

The algorithm may require additional variables for temporary storage or bookkeeping purposes, such as counters, flags, or control variables. The memory size for these variables will depend on the specific implementation and requirements.

The implementation uses dynamic memory allocation or data structures, the memory usage may change dynamically during the execution of the algorithm.

To determine the exact memory requirements for a specific implementation, it is necessary to consider the specific details of the implementation, including the data types used, the number of iterations, and any additional variables or structures introduced.

These are the datas needed for the design and implementation of slow and fast division algorithms.